Efficient Optimization of Control Libraries

Debadeepta Dey, Tian Yu Liu, Boris Sofman, Drew Bagnell

CMU-RI-TR-11-20

June 2011

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **JUN 2011**	2. REPORT TYPE	3. DATES COVERED **00-00-2011 to 00-00-2011**
4. TITLE AND SUBTITLE **Efficient Optimization of Control Libraries**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,The Robotics Institute,Pittsburgh,PA,15213**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR’S ACRONYM(S)
		11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
A popular approach to high dimensional control problems in robotics uses a library of candidate ?maneuvers? or ?trajectories?[13, 28]. The library is either evaluated on a fixed number of candidate choices at runtime (e.g. path set selection for planning) or by iterating through a sequence of feasible choices until success is achieved (e.g. grasp selection). The performance of the library relies heavily on the content and order of the sequence of candidates. We propose a provably efficient method to optimize such libraries leveraging recent advances in optimizing sub-modular functions of sequences [29]. This approach is demonstrated on two important problems mobile robot navigation and manipulator grasp set selection. In the first case performance can be improved by choosing a subset of candidates which optimizes the metric under consideration (cost of traversal). In the second case performance can be optimized by minimizing the depth the list is searched before a successful candidate is found. Our method can be used in both online and batch settings with provable performance guarantees, and can be run in an anytime manner to handle real-time constraints.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**28**	

Abstract

A popular approach to high dimensional control problems in robotics uses a library of candidate "maneuvers" or "trajectories"[13, 28]. The library is either evaluated on a fixed number of candidate choices at runtime (*e.g.* path set selection for planning) or by iterating through a sequence of feasible choices until success is achieved (*e.g.* grasp selection). The performance of the library relies heavily on the content and order of the sequence of candidates. We propose a provably efficient method to optimize such libraries leveraging recent advances in optimizing sub-modular functions of sequences [29]. This approach is demonstrated on two important problems: mobile robot navigation and manipulator grasp set selection. In the first case, performance can be improved by choosing a subset of candidates which optimizes the metric under consideration (cost of traversal). In the second case, performance can be optimized by minimizing the depth the list is searched before a successful candidate is found. Our method can be used in both online and batch settings with provable performance guarantees, and can be run in an anytime manner to handle real-time constraints.

1 Introduction

Many approaches to high dimensional robotics control problems such as grasp selection for manipulation [4, 15] and trajectory set generation for autonomous mobile robot navigation [16] use a library of candidate "maneuvers" or "trajectories". Such libraries effectively discretize a large control space and enable tasks to be completed with reasonable performance while still respecting computational constraints. The library is used by evaluating a fixed number of candidate maneuvers at runtime or iterating through a sequence of choices until success is achieved. Performance of the library depends heavily on the content and order of the sequence of candidates.

This class of problems can be framed as *list optimization* problems where the ordering of the list heavily influences both performance and computation time [27]. In such settings queries arrive sequentially and decisions have to be taken at each time step either by searching a list until success is achieved (grasp selection, novelty detection) or generating a subset of the list to be evaluated (trajectory set generation). For problems such as grasp selection where the system is searching for the first successful grasp in a list of candidate grasps, performance is dependent on the depth in the list that has to be searched before a successful answer can be found. For problems where the system must generate a subset of a bigger list to be evaluated, performance is dependent on the subset maximizing a metric. For the case of trajectory set generation this corresponds to coming up with a set of trajectories such that the computed cost of traversal of the robot is minimized.

As we show below, maneuver library optimization problems exhibit the property of *monotone sequence submodularity* [14, 29]: the value of adding each addition element to the list yields diminishing returns over earlier additions. For example, in the case of grasp selection, adding a candidate grasp to a pre-existing large list of grasps does not increase the chance of selecting a successful grasp as much as adding the candidate grasp to a smaller subsequence of that list would.

In this paper we take advantage of recent advances in submodular sequence function optimization [29] to propose an approach to high-dimensional robotics control problems that leverages the online and submodular nature of list optimization. The results of Streeter et al. [29] establish algorithms that are near-optimal (within known NP-hard approximation bounds) in both a fixed design and no-regret sense. Such results may be somewhat unsatisfactory for the control problem we address as we are concerned about performance on future data and thus we consider two such batch settings: static optimality where we consider a distribution over training examples that are independently and identically distributed (i.i.d) (grasp selection) and a form of dynamic optimality where the distribution of examples is influenced by the execution of the control libraries. We show that online-to-batch conversions [6] combined with the advances in online submodular function maximization enable us to effectively optimize these control libraries.

For the trajectory sequence selection problem, we show that our approach exceeds the performance of the current state of the art by achieving lower cost of traversal in a real-world path planning scenario [16]. For grasp selection (related to the MIN-SUM SUBMODULAR COVER problem) we show that we can adaptively reorganize a list of grasps such that the depth traversed in the list until a successful grasp is found is minimized. Our approach outperforms approaches such as random grasp orderings or orderings that rank grasps by average rate of success. We emphasize that although our approach in both cases is online in nature, it can operate in an offline mode where the system is trained using prior collected data to learn to optimize a list of candidate grasps or trajectories. During runtime the learned static list (or distribution over lists) can then be utilized for all queries without incorporating additional performance feedback.

In Section 2 we briefly review the use of control libraries for various robotics problems. In Section 3 we review the concept of sequence submodularity and outline the approach for online monotone sequence submodular function optimization we use in the two application domains. Section 4 and Section 5 explains the versions of the approach specifically applied to the two domains, experimental set-up and performance results compared with the state-of-the-art in the domains of path planning and grasp selection. We conclude in Section 6 with future work in this area.

2 Control Libraries

Control libraries approximate a large (possibly infinite) set of feasible controllers by sampling and storing a (relatively) small number of controllers. At runtime, the goal becomes to find the "best" element of the library to execute from that library. If the library was chosen well, an approximation to the optimal control can be found in the library while maintaining a limit on computation.

Stolle et al [28] have used trajectory libraries from expert demonstration to find suitable control policies in high dimensional spaces for a walking robot in an anytime manner. Frazzoli et al [13] recorded expert human pilots aggressively flying small unmanned vehicles and created a library of trajectory primitives. Given a planning task, a feasible trajectory could then be quickly generated using a concatenation of these stored trajectories. The advantage of such a library was that each of the stored trajectory primitives is guaranteed to be dynamically feasible and hence a new trajectory generated by a concatenation of such primitives is also dynamically feasible, subject to certain transition constraints.

Grasp sequence ranking is usually accomplished by evaluating the force closure and stability criterion for all grasps within a library, then executing the one with the highest score[4, 26, 7, 8]. Goldfeder et al.[15] store a library of precomputed grasps for a wide variety of objects. Given a novel object they find the closest object in the library and use the grasps associated with that object to suggest a grasp for the new scenario.

In path planning for mobile robot navigation, one of the most powerful methods leverages a "local planner" [17] that evaluates a sequence of trajectories[16] to identify the best trajectory amongst these and then advances a portion of this trajectory. This is executed in a receding-horizon fashion. Various methods have been proposed for generating suitable sequences of trajectories offline [16, 5, 10], but because in many cases this entire set of stored trajectories maybe evaluated for each situation, the size of this library is strictly limited by available online computation.

A fundamental question remaining is how such control libraries should be constructed and organized in order to maximize the performance on the task at hand while minimizing search time. We provide a data-driven approach to the problem of constructing and optimizing control libraries.

3 Review of Submodularity and Maximization of Submodular functions

A function $f : \mathscr{S} \rightarrow [0,1]$ is monotone submodular for any sequence $S \in \mathscr{S}$ where $\mathscr{S}$ is the set of all sequences if it satisfies the following two properties:

- (Monoticity) for any sequence $S_1, S_2 \in \mathscr{S}$, $f(S_1) \leq f(S_1 \cup S_2)$ and $f(S_2) \leq f(S_1 \cup S_2)$
- (Submodularity) for any sequence $S_1, S_2 \in \mathscr{S}$, $f(S_1)$ and any action $a \in \mathscr{V} \times \mathbb{R}_{>0}$, $f(S_1 \cup S_2 \cup \langle a \rangle) - f(S_1 \cup S_2) \leq f(S_1 \cup \langle a \rangle) - f(S_1)$

where $\cup$ means order dependent concatenation of lists.

In the online setting α-regret is defined as the difference in the performance of an algorithm and α times the performance of the best expert in retrospect. Streeter et al. [29] provide algorithms for maximization of submodular functions whose α-regret (regret with respect to proven NP-hard bounds) approaches zero as a function of time.

We review here the relevant parts of the online submodular function maximization approach as detailed by [29]. Assume we have a list of feasible control actions $\mathscr{A}$, a sequence of tasks $f_{1 \dots T}$, and a list of actions of length N that we maintain and present for each task. One of the key components of this approach makes use of the idea of an *expert algorithm*. In this approach, the order of the selected list for each task is chosen by N expert algorithms, each of whom gives out a piece of advice for its assigned slot in the list. The algorithm runs N distinct copies of this expert algorithm: $\mathscr{E}_1, \mathscr{E}_2, \dots, \mathscr{E}_N$, where each expert algorithm $\mathscr{E}_i$ maintains a distribution over the set of possible experts (in this case action choices). Just after task f_t arrives and before the correct sequence of actions to take for this task is shown, each expert algorithm $\mathscr{E}_i$ selects a control action a_i^t. The list order used on task f_t is then $S_t = \{a_1^t, a_2^t, \dots, a_N^t\}$. At the end of step t, the value of the reward x_i^t for each expert i is made public and is used to update each expert accordingly.

When it is possible to evaluate the marginal reward for each expert (action/control primitive) for every slot the randomized weighted majority (WMR) [22] may serve as the experts algorithm subroutine as it needs full information feedback. In the trajectory selection case one would have to evaluate all the feasible trajectories for the robot. This is an option in the offline case when time is not a constraint but is not feasible online. It is hence desirable to use an experts algorithm subroutine which requires the marginal rewards of only those actions which are chosen to populate the sequence (EXP3 [2]).

4 Application: Mobile robot navigation

Traditionally, path planning for mobile robot navigation is done in a hierarchical manner with a global planner at the top level driving the robot in the general direction of the goal while a local planner makes sure that the robot avoids obstacles while making progress towards the goal. The local planner runs at a high frequency and at every time step evaluates a set of feasible control trajectories on the immediate perceived environment to find the trajectory yielding the least cost of traversal. The robot then moves along the trajectory, which has the least sum of cost of traversal and cost to go to the goal from the end of the trajectory for one time step. This process is then repeated at each time step.

This set of feasible trajectories is usually computed offline by sampling from a much larger (possibly infinite) set of feasible trajectories. Such library-based model predictive approaches are widely used in state-of-the-art systems leveraged by most DARPA Urban Challenge, Grand Challenge (including the two highest placing teams for both)[32, 23, 31, 30] as well as on sophisticated outdoor vehicles (LAGR[18], UPI[3], Perceptor[19]) developed in the last decade. A particularly effective method for generating such a library is to generate the set of trajectories greedily such that the area between the trajectories is maximized [16]. As this method runs offline, it does not adapt to changing conditions in the environment nor is it data-driven to perform well on the environments encountered in practice.

Let $cost(a_i)$ be the cost of traversing along trajectory a_i sampled from the possible set of trajectories. Let N be the budgeted number of trajectories that can be evaluated during real-time operation. For a given set of trajectories $\{a_1, a_2, ..., a_N\}$ sampled from the set of all feasible trajectories, we define the monotone, submodular function that we maximize using the lowest-cost path from the set of possible trajectories as $f : \mathscr{S} \to [0,1]$:

$$f \equiv \frac{N_o - min(cost(a_1), cost(a_2), \ldots, cost(a_N))}{N_o} \tag{1}$$

where N_o is a constant normalizer which is the highest cost trajectory that can be expected for a given cost map.

We present the general algorithm for online selection of action sequences in Algorithm.1. The inputs to the algorithm are the number of action primitives N which can be evaluated at runtime within the computational constraints and N copies of experts algorithms, $\mathscr{E}_1, \mathscr{E}_2, ..., \mathscr{E}_N$, one for each position of the sequence of actions desired. The experts algorithm subroutine can be either Randomized Weighted Majority (WMR)[22] (Algorithm.3) or EXP3 [2](Algorithm.2). T represents the number of planning steps the robot is expected to carry out. In lines.1-5 a sequence of trajectories is sampled from the current distribution of weights over trajectories maintained by each copy of the expert algorithm using the function. Function $sampleActionExperts(\mathscr{E}_i)$ samples the distribution of weights over experts (trajectories) maintained by experts algorithm copy $\mathscr{E}_i$ to fill in slot i (S_t^i) of the sequence without repeating trajectories selected for slots before the i^{th} slot.

The sequence of trajectories S^t is evaluated on the current environment around the robot in line.6 to find the trajectory a^* which has the least sum of cost of traversal and cost to go to the goal from the end of the trajectory. This trajectory is then traversed for the time $\triangle t$ until the next planning cycle.

As a consequence of traveling the best trajectory a^* the robot encounters the next environment **ENV** (line.7). In lines.8-13 each of the experts algorithms weights over all feasible trajectories are increased if the monotone submodular function f_t is increased by adding trajectory a_j^i at the i^{th} slot.

The function *sampleActionExperts* in the case of EXP3 corresponds to executing lines.1-2 of Algorithm.2. For WMR this corresponds to executing line.1 of Algorithm.3. Similarly the function *updateWeight* corresponds to executing lines.3-6 of Algorithm.2 or lines.3-4 of Algorithm.3.

The learning rate ε for WMR is set to be $\frac{1}{\sqrt{T}}$ where T is the number of planning cycles, possibly infinite. For infinite or unknown planning time this can be set to $\frac{1}{\sqrt{t}}$ where t is the current time step. Similarly the mixing parameter γ for EXP3 is set as $min\left\{1, \sqrt{\frac{|\mathscr{A}|\ln|\mathscr{A}|}{(e-1)T}}\right\}$.

Note that actions are generic and in the case of mobile robot navigation are trajectory primitives from the control library. Later on in Section.5 actions are grasps that the manipulator can execute.

T can be possibly infinite as a ground robot can be run for abitrary amounts of time with new goals or waypoints presented to the robot every time the current goal is achieved. Since the choice of T influences the learning rate of the approach it is necessary to account for the possibility of

T being infinite.

As mentioned in Section.3 WMR may be too computationally costly for *online* applications as it requires the evaluation of every trajectory at every point in the list whether executed or not. EXP3, by contrast, learns more slowly but requires as feedback only the cost of the sequence of trajectories actually executed, and hence add negligible overhead on the trajectory library approach. For EXP3 line.9 would loop over only the experts chosen at the current time step instead of $|\mathscr{A}|$.

We refer to this sequence optimization algorithm in the rest of the paper as SEQOPT.

Algorithm 1 Algorithm for trajectory sequence selection

```
Require: number of trajectories $N$, experts algorithms subroutine copies (Algorithms.2 and 3) $\mathscr{E}_1, \mathscr{E}_2, \ldots, \mathscr{E}_N$
 1: for $t = 1$ to $T$ do
 2:     for $i = 1$ to $N$ do
 3:         $a_i = sampleActionExperts(\mathscr{E}_i)$
 4:         $S_t^i \leftarrow a_i$
 5:     end for
 6:     $a^* = evaluateActionSequence(\mathbf{ENV}, S_t)$
 7:     $\mathbf{ENV} = getNextEnvironment(a^*, \triangle t)$
 8:     for $i = 1$ to $N$ do
 9:         for $j = 1$ to $|\mathscr{A}|$ do
10:             $reward_j^i = f_t(S_t^{\langle i-1 \rangle} \cup a_j^i) - f_t(S_t^{\langle i-1 \rangle})$
11:             $w_j^i \leftarrow updateWeight(reward_j^i, w_j^i)$
12:         end for
13:     end for
14: end for
```

SEQOPT: the approach detailed here and inherited from [29] is an online algorithm which produces a sequence which converges to the greedy sequence as the time horizon grows. The greedy sequence is guaranteed to achieve at least $1 - 1/e$ of the value of the optimal list [11]. Therefore SEQOPT is a 0 α-regret (for $\alpha = 1 - 1/e$ here) algorithm. This implies that its α-regret goes to 0 at a rate of $O(1/\sqrt{T})$ for T interactions with the environment.

We are also interested in its performance with respect to future data and hence consider notions of near-optimality with respect to distributions of

Algorithm 2 Experts Algorithm: Exponential-weight algorithm for Exploration and Exploitation (EXP3) [2]

```
Require: $\gamma \in (0,1]$, initialization $w_j = 1$ for $j = 1, \dots, |\mathscr{A}|$
1: Set $p_j = (1-\gamma)\frac{w_j}{\sum_{j=1}^{|\mathscr{A}|} w_j} + \frac{\gamma}{|\mathscr{A}|}$ $j = 1, \dots, |\mathscr{A}|$
2: Randomly sample $i$ according to the probabilities $p_1, \dots, p_{|\mathscr{A}|}$
3: Receive $reward_j \in [0,1]$
4: for $j = 1$ to $|\mathscr{A}|$ do
5:     $$\hat{x}_t = \begin{cases} \frac{x_t}{p_j} & \text{if } t = i \\ 0 & \text{otherwise} \end{cases}$$
6:     $w_t \leftarrow w_t exp(\frac{\gamma \hat{x}_t}{|\mathscr{A}|})$
7: end for
```

Algorithm 3 Experts Algorithm: Randomized Weighted Majority (WMR) [22]

```
Require: Initialization $w_j = 1$ for $j = 1, \dots, |\mathscr{A}|$
1: Randomly sample $j$ according to the distribution of weights $w_1, \dots, w_{|\mathscr{A}|}$
2: Receive rewards for all experts $reward_1, \dots, reward_{|\mathscr{A}|}$
3: for $j = 1$ to $|\mathscr{A}|$ do
4:     $$w_j = \begin{cases} w_j(1+\varepsilon)^{reward_j} \text{ if } reward_j \geq 0 \\ w_j(1-\varepsilon)^{-reward_j} \text{ if } reward_j < 0 \end{cases}$$
5: end for
```

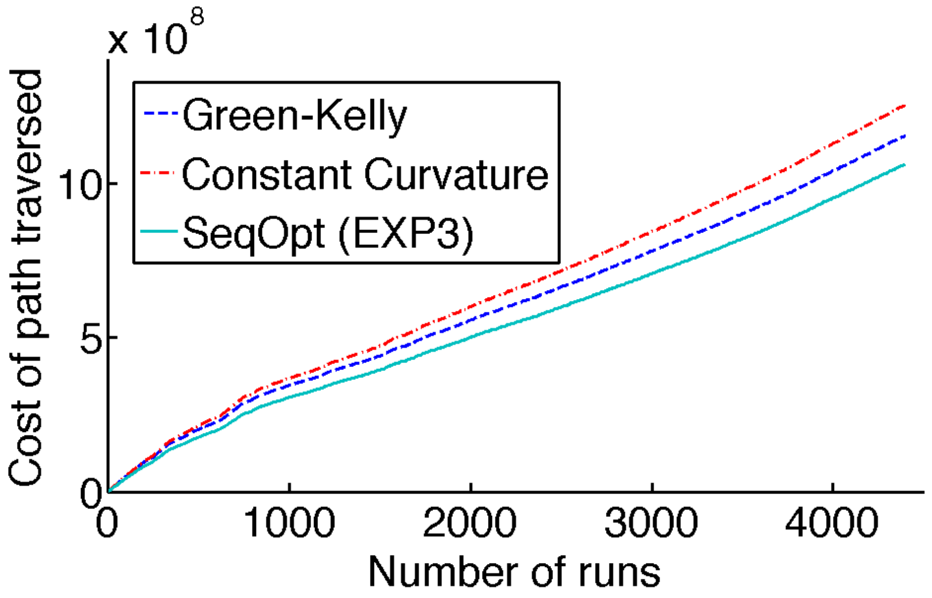


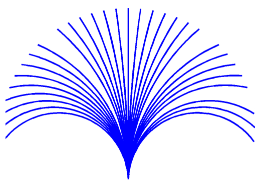
Figure 1: The cost of traversal of the robot on the cost map of Fort Hood, TX using trajectory sequences generated by different methods for 30 trajectories per time step over 1055788 planning cycles in 4396 runs. Constant curvature trajectories result in the highest cost of traversal followed by Green-Kelly path sets. Our sequence optimization approach (SEQOPT) using EXP3 as the experts algorithm subroutine results in the lowest cost of traversal (8% lower than Green-Kelly) with negligible overhead.

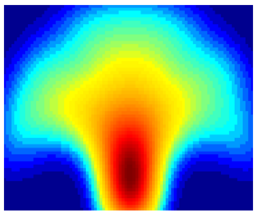
environments. We define a *statically optimal* sequence of trajectories $S_{so} \in \mathscr{S}$ as:

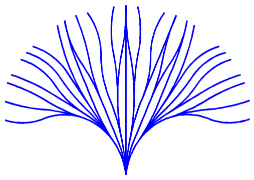
$$S_{so} = \underset{S}{\operatorname{argmax}} E_{d(\mathbf{ENV})}[f(\mathbf{ENV}, S)] \tag{2}$$

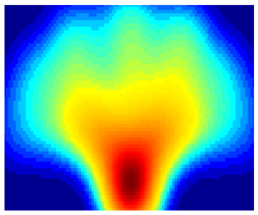
where $d(\mathbf{ENV})$ is a distribution of environments that are randomly sampled. The trajectory sequence S is evaluated at each location. A statically near-optimal trajectory sequence S_{so} thus approximately maximizes the expectation of the Equation 23 ($E_{d(\mathbf{ENV})}[f(\mathbf{ENV}, S)]$) over the distribution of environments **ENV**, effectively optimizing the one-step cost of traversal at the locations sampled from the distribution of the environments.

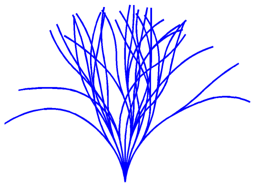
Knepper et al. [20] note that sequences of trajectories are generally designed for this kind of static planning paradigm but are used in a dynamic planning paradigm where the library choice influences the examples seen

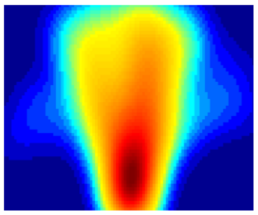

(a) Constant Curvature

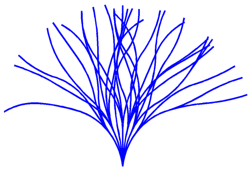

(b) Constant Curvature Density


(c) Green-Kelly


(d) Green-Kelly Density


(e) SEQOPT (EXP3) Dynamic


(f) SEQOPT (EXP3) Dynamic Density


(g) SEQOPT (EXP3) Static

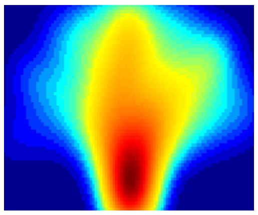
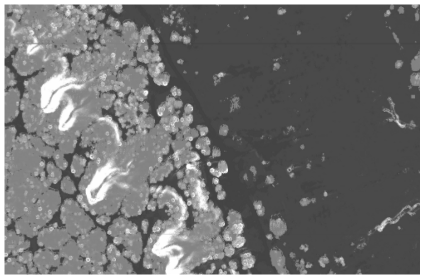

(h) SEQOPT (EXP3) Static Density

Figure 2: The density of distribution of trajectories learned by our approach (SEQOPT using EXP3) for the dynamic planning paradigm in Figure.2e shows that most of the trajectories are distributed in the front whereas for the static paradigm they are more spread out to the side. This shows that for the dynamic case more trajectories should be put in the front of the robot as obstacles are more likely to occur to the side as pointed out by Knepper et al [20]


(a) Cost map of Fort Hood, Texas

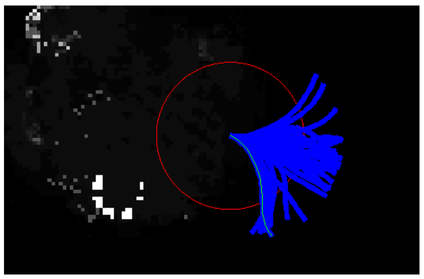

(b) Robot simulated driving over cost map towards goal

Figure 3: We used a real-world cost map of Fort Hood, Texas(3a) and simulated a robot driving over the map to random goal locations from random start locations (3b) using trajectory sequences generated by different methods for comparision. The trajectory in green is evaluated to be the least cost for the pictured planning cycle.

and that there is little correlation in performance between good static and good dynamic performance for a sequence. Our approach bridges this gap by allowing offline batch training on a fixed distribution, or allowing samples to be generated by running the currently sampled library.

We then define a *weakly dynamically optimal* trajectory sequence $S_{wdo} \in \mathscr{S}$ as:

$$S_{wdo} = \underset{S}{\operatorname{argmax}}[f(\mathbf{ENV}, S)] \tag{3}$$

where $d(\mathbf{ENV}|\pi)$ is defined as the distribution of environments that are induced by the robot following the policy π. The policy π corresponds to the robot following the least cost trajectory within S_{wdo} at each situation encountered. Hence a weakly dynamically optimal trajectory sequence minimizes the cost of traversal of the robot at all the locations which the robot encounters as a consequence of executing the policy π. We define this as weakly dynamically optimal as there can be other trajectory sequences $S \in \mathscr{S}$ that can minimize the cost of traversal with respect to the distribution of environments induced by following the policy π.

Knepper et al [20] further note the surprising fact that for a vehicle following a reasonable policy, averaged over time-steps the distribution of obstacles encountered ends up heavily weighted to the sides. Good earlier policy choices imply that the space to the immediate front of the robot is mostly devoid of obstacles. It is effectively a chicken-egg problem to find such a

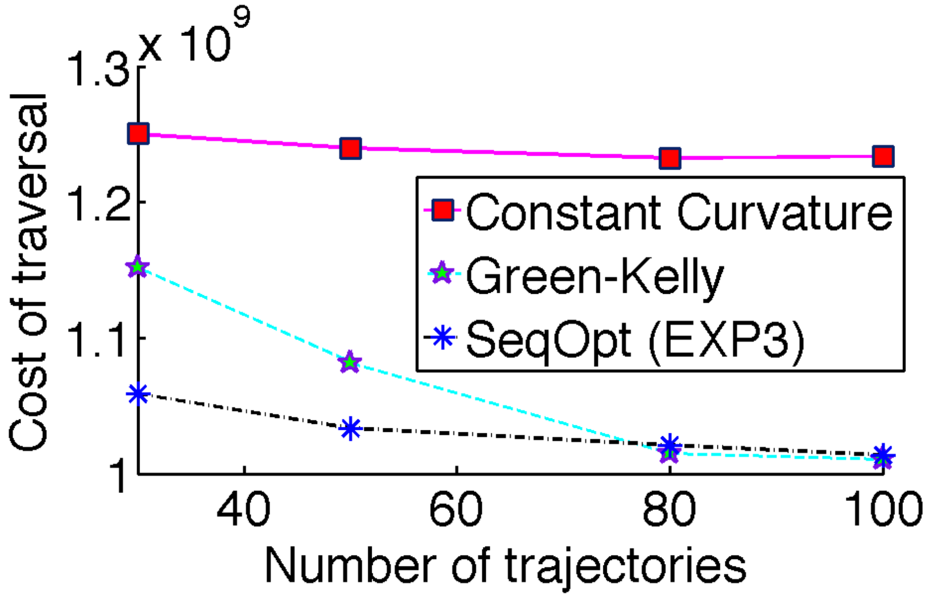


Figure 4: As the number of trajectories evaluated per planning cycle are increased the cost of traversal for trajectory sequences generated by Green-Kelly and our method drops and at 80-100 trajectories achieve almost the same cost of traversal. It is to be noted that our approach decreases the cost of traversal much faster than Green-Kelly trajectory sequences.

policy with respect to its own induced distribution of examples, which we address here as *weakly dynamically optimality*.

We briefly note the following propositions about the statistical performance of Algorithm 1. We elide full proofs to the appendix, but note that they follow from recent results of online-to-batch learning, [25] combined with the regret guarantees of [29] on the objective functions we present.

Proposition 1. (Approximate Static Optimality) If *getNextEnvironment* returns independent examples from a distribution over environments (*i.e.*, the chosen control does not affect the next sample), then for a list S chosen randomly from those generated throughout the T iterations of the Algorithm 1, it holds that $E_{d(\mathbf{ENV})}[(1-1/e)f(S*)-f(S)] \leq O(\frac{ln(1/\delta)}{\sqrt(T)})$ with probability greater then $1-\delta$.

Proposition 2. (Approximate Weak Dynamic Optimality) If *getNextEnvironment* returns examples by forward simulating beginning with a random environment and randomly choosing a new environment on reaching a goal, then consider the policy π_{mixture} that begins each new trial by choosing a list randomly from those generated throughout the T iterations of the Algorithm 1. By the no-regret property, such a mixture policy will be α approximately dynamically optimal in expectation up to an additive term $O(\frac{ln(1/\delta)}{\sqrt{T}})$ with probability greater then $1-\delta$. Further, in the (experimentally typical) case where the distribution over library sequences converges, the resulting single list is (up to approximation factor α) weakly dynamically optimal.

4.1 Experimental setup

We simulated a robot driving over a real-world cost map generated for Fort Hood, Texas (Figure.3) with trajectory sequences generated by using the method devised by Green et al. [16] for both constant curvature arcs (Figures.2a, 2b) and trajectories comprised of concatenation of arcs of different curvatures (Figures.2c, 2d). The cost map and parameters for the local planner (number of trajectories to evaluate per time step, length of the trajectories, fraction of trajectory traversed per time step) were taken to most closely match that given in Bagnell et al. [3].

4.2 Results

4.2.1 Dynamic Simulation

Figure.1 shows the cost of traversal of the robot with different trajectory sets as a function of number of runs. Each run constitutes the robot starting from a random starting location and ending at the specified goal on the map. 100 goal locations and 50 start locations for every goal location were chosen at random. The set of weights for the N copies of experts algorithm EXP3 were carried over through consecutive runs.

The cost of traversal of constant curvature trajectory sequences grows at the highest rate followed by using the Green-Kelly path set. The lowest cost of traversal is achieved by running Algorithm.1 with EXP3 as the experts algorithm subroutine. At the end of 4396 runs there is a 8% reduction in cost of traversal between Green-Kelly and our approach (SEQOPT using EXP3). It is to be emphasized that improvement in path planning is obtained with negligible overhead. Though the complexity of our approach scales linearly in the number of motion primitives and depth of the library, each operation is simply a multiplicative update and a sampling step. In practice it was not possible to evaluate even a single extra motion primitive in the time overhead that our approach requires. In 1 millisecond 100000 update steps can be performed using Exp3 as the experts algorithm subroutine.

4.2.2 Static Simulation

In addition to the dynamic simulations we also performed a static simulation where for 100 goal locations the robot was placed at 500 random poses in the cost map and the cost of traversal of the selected trajectory a^* over the next planning cycle was recorded. SEQOPT with EXP3 and Green-Kelly sequences obtained 0.5% and 0.25% lower cost of traversal than constant curvature sequences respectively. The performance for all three methods was essentially at par. This can be explained by the fact that Green-Kelly trajectory sequences are essentially designed to handle the static case of planning where trajectories must provide adequate density of coverage in all directions as the distribution of obstacles is entirely unpredictable in this case.

In the dynamic planning case on the other hand, the situations the robot encounters are highly correlated and because the robot is likely to be guided by a global trajectory, a local planner that tracks that trajectory well will likely benefit from a higher density of trajectories toward the front as most of the obstacles will be to the sides of the path. This is evident by the densities of generated trajectory sequences for each case as shown in Figure.2. Our approach naturally deals with this divide between the static and dynamic planning paradigms by adapting the chosen trajectory sequence at all times. A video demonstration of the algorithm can be found at the following link:[1]

5 Application: Grasp selection for manipulation

Most of the past work on grasp set generation and selection have focused on automatically producing a successful and stable grasp for a novel object, and the computational time is of secondary concern. As a result very few grasp selection algorithms have attempted to optimize the order of consideration in grasp databases. Berenson et al. [4] dynamically ranked pre-computed grasps by calculating a grasp-score based on force closure, robot position, and environmental clearance. Ratliff et al. [24] employed imitation learning on demonstrated example grasps to select a grasp in a discretized grasp space. In both of these cases the entire library of grasps is evaluated for each new environment or object at run time, and the order of the entries and their effect on computation are not considered. In this section we describe our grasp ranking procedure, which uses trajectory generation success to reorder a list of database grasps, so that for a majority of situation encountered in a new environment, only a subset of grasp entries near the front of the control library need to be evaluated.

For a sequence of grasps $S \in \mathscr{S}$ we define the submodular monotone grasp selection function $f : \mathscr{S} \rightarrow [0,1]$ as $f \equiv P(S)$ where $P(S)$ is the probability of successfully grasping an object in a given scenario using the sequence of grasps provided.

For a given sequence of grasps $S \in \mathscr{S}$ we want to minimize the cost of evaluating the sequence i.e. minimize the depth in the list that has to be searched until a successful grasp is found. Thus the cost of a sequence of grasps can be defined as $c = \sum_{i=0}^{N} 1 - f(S_{\langle i \rangle})$ where $f(S_{\langle i \rangle})$ is defined as the value of the submodular function f on executing sequence $S \in \mathscr{S}$ up to $\langle i \rangle$ slots in the sequence. Minimizing c corresponds to minimizing the depth i in the sequence of grasps that must be evaluated for a successful grasp to be found. (We assume that every grasp takes equal time to evaluate)

The same algorithm for trajectory sequence generation (Algorithm.1) is used here grasp sequence generation. Here the set of experts for each experts algorithm are the set of grasps in the grasp library. Here each experts algorithm $\mathscr{E}_i$ maintains a set of weights for each grasp (expert) in the library. A sequence of grasps is constructed by sampling without repetition the distribution of weights for each grasp $\mathscr{E}_i$ for each position i in the sequence (lines:1-5). This sequence is evaluated on the current environment until a successful grasp a^* is found (line:6). Not that the next environment is then presented to the robot from a distribution of environments and is not obtained by evaluating the successful grasp unlike the path planning case. If the sucessful grasp was found at position i in the sequence then in experts algorithm $\mathscr{E}_i$ the weight corresponding to the successful grasp id is updated using SEQOPT with EXP3's update rule. For WMR all the grasps in the sequence are evaluated and the weights for every expert are updated according to lines.9-12.

5.1 Experimental setup

We performed experiments using environments containing a trigger-style flashlight as the target object. We used the OpenRAVE[9] simulation framework to generate a multitude of different grasps and environments for each object. The manipulator used in this experiment is a Barret WAM arm and

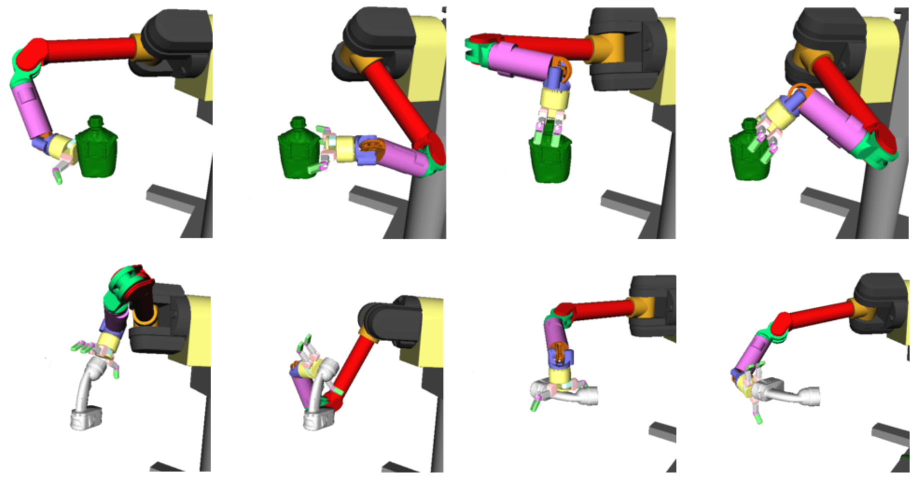


Figure 5: Example grasps from the grasp library sequence. Each grasp has a different approach direction and finger joint configuration recorded with respect to the object's frame of reference. Our algorithm attempts to re-order the grasp sequence to quickly cover the space possible scenarios with fewer grasps at the front of the sequence.

hand with a fixed base, and a 3D joystick is used to control the simulated robot in grasp sequence generation. Since the grasps are generated by a human operator, we assume they are stable grasps and hence the main failure mode is in trajectory planning and obstacle collision. During both training and testing, bidirectional RRT [21] is used to generate the trajectory from the manipulator's current position to the target grasp position.

The grasp library consisted of 60 grasps and the library was evaluated on 50 different environments for training, and 50 environments for testing. For a particular environment/grasp pair the grasp success is evaluated by the success of Bi-RRT trajectory generation, and the grasp sequence ordering is updated at each timestep of training. For testing and during run-time, the learned list was evaluated without further feedback.

We compared the performance of SEQOPT with EXP3 as well as WMR as expert subroutines to two methods of grasp library ordering: a random grasp ordering, and an ordering of the grasps by decreasing success rate across all examples in training (which we will call "frequency"). At each time step of the training process, a random environment was selected from the training set and each of the four grasp sequence orderings were evaluated. The search depth for each test case was tracked to compute overall

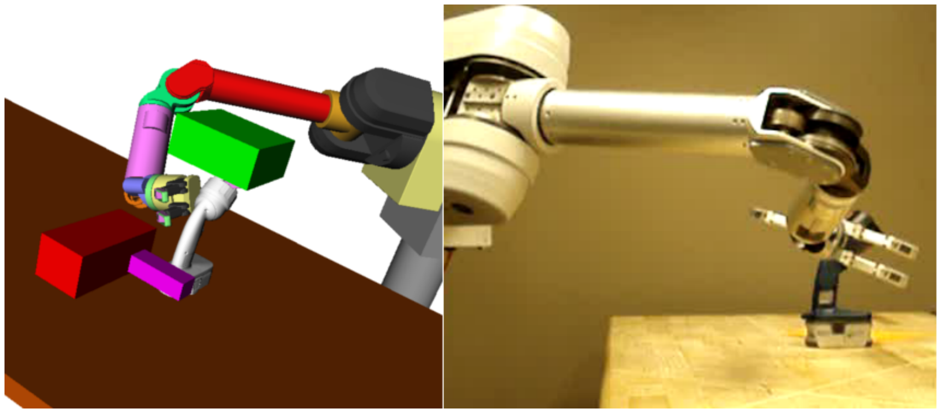


Figure 6: Executing a particular grasp in both simulation and real hardware. The grasp library ordering is trained in simulation only, and the resulting grasp sequence can be executed in hardware without modifications.

performance. The performance of the two naive ordering methods does not improve over time because the frequency method is a single static sequence and the random method has a uniform distribution over all possible rankings.

5.2 Results

The performance of each sequence after training is shown in Figure 7. We can clearly see a dramatic improvement in the performance of SEQOPT run with both WMR and EXP3 update rules over the random and frequency methods. While random and frequencymethods produce a grasp sequence ordering that requires an average of about 7 evaluations before a successful grasp is found, SEQOPT with WMR and EXP3 produce a more optimized ordering that require only about 5 evaluations which is 29% improvement. Since evaluating a grasp entails planning to the goal and executing the actual grasp this improvement is significant speedup in finding a successful grasp. Again this improvement comes at negligible cost and in practice it wasnt possible to evaluate a single extra grasp in the extra time overhead for our approach.

It is interesting to note that a random ordering of the grasps has similar performance to the frequency method. This is because similar grasps tend to be correlated in their success and failure, so the grasps in the front of the frequency ordering tend to be similar. When the first grasp fails, the next few are likely to fail as well, increasing the average search depth. The

SEQOPT algorithm solves this correlation problem by ordering the grasp sequence such that the grasps near the front of the library cover the space of possible configurations as quickly as possible. A video demonstration of the algorithm can be found at the following link:[1]

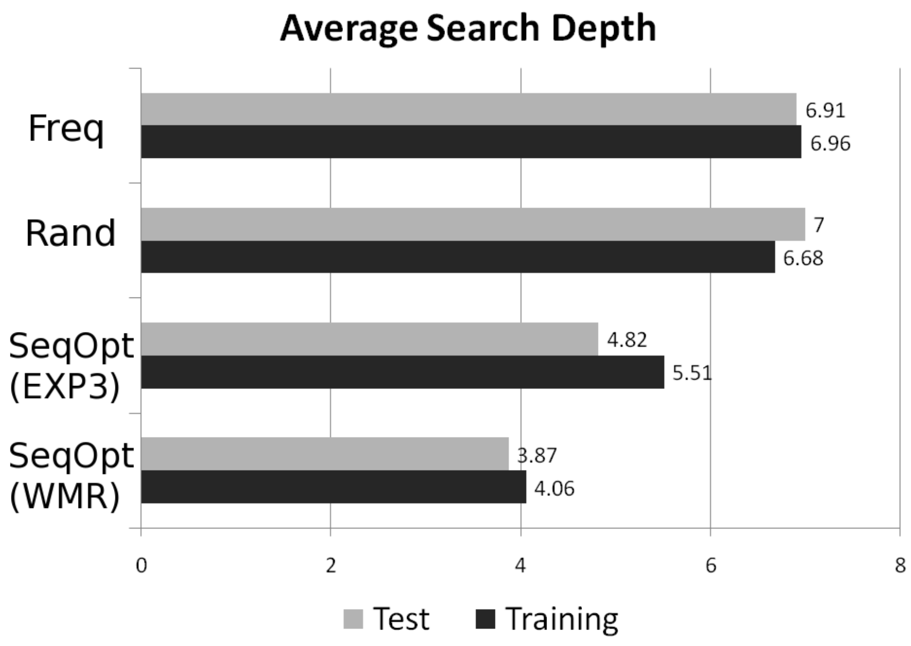


Figure 7: Average depth till successful grasp for flashlight object with 50 test environments. The training data shows the average search depth achieved at the end of the training session over 50 training environments. Algorithm.1 (SEQOPT) when run with EXP3 as the experts algorithm subroutine achieves 20% reduction over grasp sequences arranged by average rate of success (Freq.) or a random ordering of the grasp list (Rand.)

6 Conclusion

We have shown an efficient method for optimizing performance of control libraries and have attempted to answer the question of how to contruct and order such libraries.

The grasp sequence generation method presented here does not incorporate the context in which the object is placed in the current environment. We aim to modify the current approach to close the loop with perception and take account of features in the environment for grasp sequence generation.

As robots employ increasingly large control libraries to deal with the diversity and complexity of environments that they may encounter, approaches such as the ones presented here will become crucial to maintaining robust real-time operation.

7 Acknowledgements

This work was funded by the Army Research Laboratories through the Robotics Collaborative Technology Alliance (R-CTA) and the Defense Advanced Research Projects Agency through the Autonomous Robotic Manipulation Software Track (ARM-S).

References

[1] URL `http://youtube.com/robotcontrol1`.

[2] P. Auer, N. Cesa-Bianchi, Y. Freund, and R.E. Schapire. The non-stochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2003.

[3] J.A. Bagnell, D. Bradley, D. Silver, B. Sofman, and A. Stentz. Learning for autonomous navigation. *Robotics Automation Magazine, IEEE*, 17(2):74 –84, june 2010.

[4] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J Kuffner. Grasp planning in complex scenes. In *IEEE-RAS Humanoids*, December 2007.

[5] M.S. Branicky, R.A. Knepper, and J.J. Kuffner. Path and trajectory diversity: Theory and algorithms. In *ICRA*, pages 1359–1364. IEEE, 2008.

[6] N. Cesa-Bianchi, A. Conconi, and C. Gentile. On the generalization ability of on-line learning algorithms. *Information Theory, IEEE Transactions on*, 50(9):2050 – 2057, 2004.

[7] E. Chinellato, R.B. Fisher, A. Morales, and A.P. del Pobil. Ranking planar grasp configurations for a three-finger hand. In *ICRA*, volume 1, pages 1133–1138. IEEE, 2003.

[8] M.T. Ciocarlie and P.K. Allen. On-line interactive dexterous grasping. In *EuroHaptics*, page 104, 2008.

[9] R Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[10] L.H. Erickson and S.M. LaValle. Survivability: Measuring and ensuring path diversity. In *ICRA*, pages 2068–2073. IEEE, 2009.

[11] U. Feige. A threshold of ln n for approximating set cover. *JACM*, 45 (4):634–652, 1998.

[12] U. Feige, L. Lovász, and P. Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.

[13] E. Frazzoli, MA Dahleh, and E. Feron. Robust hybrid control for autonomous vehicle motion planning. In *Decision and Control, 2000.*, volume 1, 2000.

[14] S. Fujishige. *Submodular functions and optimization*. Elsevier Science Ltd, 2005.

[15] C. Goldfeder, M. Ciocarlie, J. Peretzman, H. Dang, and P.K. Allen. Data-driven grasping with partial sensor data. In *IROS*, pages 1278–1283. IEEE, 2009.

[16] C. Green and A. Kelly. Optimal sampling in the space of paths: Preliminary results. Technical Report CMU-RI-TR-06-51, Robotics Institute, Pittsburgh, PA, November 2006.

[17] T. Howard, C. Green, and A. Kelly. State space sampling of feasible motions for high performance mobile robot navigation in highly constrained environments. In *FSR*, July 2007.

[18] LD Jackel et al. The DARPA LAGR program: Goals, challenges, methodology, and phase I results. *JFR*, 23(11-12):945–973, 2006.

[19] Alonzo Kelly et al. Toward reliable off road autonomous vehicles operating in challenging environments. *IJRR*, 25(1):449–483, May 2006.

[20] R. Knepper and M.T. Mason. Path diversity is only part of the problem. In *ICRA*, May 2009.

[21] Jr. Kuffner, J.J. and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *ICRA*, volume 2, pages 995 –1001, 2000.

[22] N. Littlestone and M.K. Warmuth. The Weighted Majority Algorithm. *INFORMATION AND COMPUTATION*, 108:212–261, 1994.

[23] M. Montemerlo et al. Junior: The stanford entry in the urban challenge. *JFR*, 25(9):569–597, 2008.

[24] N. Ratliff, J.A. Bagnell, and S. Srinivasa. Imitation learning for locomotion and manipulation. Technical Report CMU-RI-TR-07-45, Robotics Institute, Pittsburgh, PA, December 2007.

[25] S. Ross, G.J. Gordon, and J.A. Bagnell. No-Regret Reductions for Imitation Learning and Structured Prediction. *Arxiv preprint arXiv:1011.0686*, 2010.

[26] J.P. Saut and D. Sidobre. Efficient Models for Grasp Planning With A Multi-fingered Hand. In *Workshop on Grasp Planning and Task Learning by Imitation*, volume 2010, 1918.

[27] B. Sofman, J.A. Bagnell, and A. Stentz. Anytime online novelty detection for vehicle safeguarding. In *ICRA*, May 2010.

[28] M. Stolle and C.G. Atkeson. Policies based on trajectory libraries. In *ICRA*, pages 3344–3349. IEEE, 2006.

[29] M. Streeter and D. Golovin. An online algorithm for maximizing submodular functions. In *NIPS*, pages 1577–1584, 2008.

[30] Sebastian Thrun et al. Stanley: The robot that won the darpa grand challenge: Research articles. *J. Robot. Syst.*, 23:661–692, September 2006.

[31] Christopher Urmson et al. A robust approach to high-speed navigation for unrehearsed desert terrain. *JFR*, 23(1):467–508, August 2006.

[32] Christopher Urmson et al. Autonomous driving in urban environments: Boss and the urban challenge. *JFR*, 25(1):425–466, June 2008.

A Proof of monotone submodularity

A sequence function f which maps subsequences $\mathscr{A} \subseteq \mathscr{V}$ of a finite sequence $\mathscr{V}$ to the real numbers. f is called submodular if, for all $\mathscr{A} \subseteq \mathscr{B} \subseteq \mathscr{V}$ and $\mathscr{S} \in \mathscr{V} \setminus \mathscr{B}$ it holds that

$$f(\mathscr{A} \oplus \mathscr{S}) - f(\mathscr{A}) \geq f(\mathscr{B} \oplus \mathscr{S}) - f(\mathscr{B}) \tag{21}$$

where $\oplus$ is the concatenation operator. Such a function is monotone if it holds that for any sequences $\mathscr{S}_1, \mathscr{S}_2 \in \mathscr{V}$, we have

$$\begin{aligned} f(\mathscr{S}_1) &\leq f(\mathscr{S}_1 \oplus \mathscr{S}_2) \\ f(\mathscr{S}_2) &\leq f(\mathscr{S}_1 \oplus \mathscr{S}_2) \end{aligned} \tag{22}$$

For the trajectory selection case we want to prove that f (in equation 1 restated here) is monotone, submodular.

$$f \equiv \frac{N_o - \min_{a_i \in A}(cost(a_i))}{N_o} \tag{23}$$

where A is the set of all feasible actions or control sequences. This can be proved if $\min_{a_i \in A}(cost(a_i))$ is monotone,supermodular. A function f is supermodular if it holds that

$$f(\mathscr{A} \oplus \mathscr{S}) - f(\mathscr{A}) \leq f(\mathscr{B} \oplus \mathscr{S}) - f(\mathscr{B}) \tag{24}$$

Theorem 1. *The function* $\min_{a_i \in A}(cost(a_i))$ *is monotone, supermodular where* a_i *are trajectories sampled from the set of all feasible trajectories.*

Proof. **Submodularity**

Assume that we are given sequences $\mathscr{A} \subseteq \mathscr{B} \subseteq \mathscr{V}$, $\mathscr{S} \in \mathscr{V} \setminus \mathscr{B}$. We want to prove the inequality in equation.24. Let $\mathscr{R} = \mathscr{B} \setminus \mathscr{A}$, the set of elements that are in $\mathscr{B}$ but not in $\mathscr{A}$. Since $\mathscr{A} \oplus \mathscr{R} = \mathscr{B}$ we can now rewrite equation.24 as

$$f(\mathscr{A} \oplus \mathscr{S}) - f(\mathscr{A}) \leq f(\mathscr{A} \oplus \mathscr{R} \oplus \mathscr{S}) - f(\mathscr{A} \oplus \mathscr{R}) \tag{25}$$

We refer to the left and right sides of equation.25 as LHS and RHS respectively. Define a^* as the trajectory which has the least cost when evaluated on a given environment. Hence there can be three cases:

- Case 1: $a* \in \mathscr{A}$ In this case $LHS = RHS = 0$
- Case 2: $a* \in \mathscr{R}$ In this case $RHS \geq LHS$

- Case 3: $a* \in \mathscr{S}$ In this case $RHS \geq LHS$

Since in all possible cases it can be seen that RHS is greater than or equal to LHS it is proved that $\min_{a_i \in A}(cost(a_i))$ is supermodular. Note that if there are multiple trajectories which have the same minimum cost as a^* then similar arguments still hold and in the worst case when they are distributed across $\mathscr{S}$,$\mathscr{R}$,$\mathscr{A}$ Case 1 holds.

Monotonicity Consider two sequences $\mathscr{S}_\infty$ and $\mathscr{S}_\in$. Define a^* as the trajectory which has the least cost when evaluated on a given environment. We want to prove that $\min_{a_i \in A}(cost(a_i))$ is monotone decreasing, ie.

$$f(\mathscr{S}_1) \geq f(\mathscr{S}_1 \oplus \mathscr{S}_2) \\ f(\mathscr{S}_2) \geq f(\mathscr{S}_1 \oplus \mathscr{S}_2) \tag{26}$$

Hence there can be three cases:

- Case 1: $a^* \in \mathscr{S}_1 \implies f(\mathscr{S}_1) = f(\mathscr{S}_1 \oplus \mathscr{S}_2)$ and $f(\mathscr{S}_2) \geq f(\mathscr{S}_1 \oplus \mathscr{S}_2)$
- Case 2: $a^* \in \mathscr{S}_2 \implies f(\mathscr{S}_1) \geq f(\mathscr{S}_1 \oplus \mathscr{S}_2)$ and $f(\mathscr{S}_2) = f(\mathscr{S}_1 \oplus \mathscr{S}_2)$
- Case 3: $a^* \in \mathscr{S}_1 \oplus \mathscr{S}_2 \implies f(\mathscr{S}_1) = f(\mathscr{S}_1 \oplus \mathscr{S}_2)$ and $f(\mathscr{S}_2) = f(\mathscr{S}_1 \oplus \mathscr{S}_2)$

Since in all possible cases the conditions in equation.26 are satisfied $\min_{a_i \in A}(cost(a_i))$ is monotone decreasing. □

Corollary 1. *The function f in equation.23 in the paper is monotone, submodular due to $\min_{a_i \in A}(cost(a_i))$ being monotone, supermodular by Theorem.1.*

□

B Background

Following a similar analysis to Ross et al. [25] We define the loss function as the difference between the maximization of $f_{\mathbf{ENV}}$ that is achieved by executing the sequence S and that achieved by executing the greedy sequence $(1-1/e)f_{\mathbf{ENV}}(S*)$

$$l(\mathbf{ENV}, S) = [(1-1/e)f_{\mathbf{ENV}}(S*) - f_{\mathbf{ENV}}(S)] \tag{27}$$

Here S^* is the best sequence that maximizes $f_{\mathbf{ENV}}$. The $(1-1/e)$ term is due to the fact that we are competing with respect to the greedy sequence and not the best sequence. This is because finding the best sequence has been proven to be NP-hard but the greedy sequence has the property that it achieves performance that is at least $(1-1/e)$ (63%) of the best sequence by Feige et al. [12].

A no-regret algorithm produces a sequence of policies π_1, π_2, ..., π_T such that the regret with respect to the best policy in hindsight goes to 0 as T goes to ∞. In the case of SEQOPT the policy at a time step corresponds to the sequence of trajectories which are evaluated and the minimum cost trajectory executed till the next time step. SEQOPT is a no α-regret algorithm which converges to the greedy list and hence $\alpha = (1-1/e)$ for SEQOPT. This can be formalized as:

$$\frac{1}{T}\sum_{i=1}^{T} l_i(S_i) - \min_{s\in\mathscr{S}} \frac{1}{T}\sum l_i(S) \leq \gamma_T \tag{28}$$

for $lim_{T\to\infty}\gamma_T = 0$ We choose the loss functions to be the expected loss under the distribution of environments, $l_i(S) = E_{d(\mathbf{ENV})}[l(\mathbf{ENV}_i, S)]$ We also define $l_T^* = \min_{s\in\mathscr{S}} \frac{1}{T}\sum_{i=1}^{T} E_{d(\mathbf{ENV})}[l(\mathbf{ENV}, S)]$ as the loss of the best policy in hindsight after T iterations.

C Proof of Proposition 1

Proposition 1. (Approximate Static Optimality) If *getNextEnvironment* returns independent examples from a distribution over environments (*i.e.*, the chosen control does not affect the next sample), then for a list S chosen randomly from those generated throughout the T iterations of the Algorithm 1, it holds that $E_{d(\mathbf{ENV})}[(1-1/e)f_{\mathbf{ENV}}(S*) - f_{\mathbf{ENV}}(S)] \leq O(\sqrt{\frac{ln(1/\delta)}{T}})$ with probability greater then $1-\delta$.

Proof. If we can evaluate the expectation over the distribution of environments exactly (infinite number of samples) then we have the following theorem:

Theorem 2. *For* SEQOPT, *for the case when environments are independently sampled from a distribution of environments there exists a sequence* $S \in S_{1:T}$ *s.t.* $E_{d(\mathbf{ENV})}[l(\mathbf{ENV}, S)] \leq l_T^* + \gamma_T$ *when the expectation over the distribution of environments can be exactly evaluated.*

Proof.

$$\begin{aligned}
& \min_{S \in S_{1:T}} E_{d(\mathbf{ENV})}[l(\mathbf{ENV}, S)] && (29)\\
\leq\ & \frac{1}{T}\sum_{i=1}^{T} E_{d(\mathbf{ENV})}[l(\mathbf{ENV}, S)] \\
\leq\ & \gamma_T + \min_{S \in \mathscr{S}} \frac{1}{T}\sum_{i=1}^{T} l_i(S) \quad (\textit{no regret}) \\
\leq\ & \gamma_T + l_T^*
\end{aligned}$$

□

The previous results hold if the online learning algorithm observes the infinite sample loss, i.e. the loss on the true distribution of environments. In practice however the algorithm would only observe its loss on a small sample of environments. We wish to bound the true loss under the distribution of environments as a function of the regret on the finite sample of environments.

If we assume that SEQOPT samples m environments in every one of the T iterations then and then observes the loss $l_i(S) = E_{d(\mathbf{ENV})}[l(\mathbf{ENV}, S)]$ for D_i, the dataset of these m environments. We restate the regret definition using this finite dataset as $\frac{1}{T}\sum_{i=1}^{T} E_{D_i(\mathbf{ENV})}[l(\mathbf{ENV}, S_i)] - \min_{s \in \mathscr{S}} \frac{1}{T}\sum_{i=1}^{T} \leq \gamma_T E_{D_i(\mathbf{ENV})}[l(\mathbf{ENV}, S)] \leq \gamma_T$. Let $\hat{l}_T^* = \min_{s \in \mathscr{S}} \frac{1}{T}\sum_{i=1}^{T} E_{D_i(\mathbf{ENV})}[l(\mathbf{ENV}, S)]$ be the training loss of the best policy in hindsight. Then we have for the finite sample case the following theorem:

Theorem 3. *For* SEQOPT*, with probability at least* $1-\delta$*, there exists a policy* $S \in S_{1:T}$ *s.t.* $E_{d(\mathbf{ENV}}[l(\mathbf{ENV}, S)] \leq \hat{l}_T^* + \gamma_T + l_{max}\sqrt{\frac{2ln(\frac{1}{\delta})}{mT}}$ *for the case when environments are independently sampled from a distribution of environments, when the distribution of environments is sampled m times.*

Proof. Let Y_{ij} be the difference between the expected per step loss of S_i under the environment distribution $d(\mathbf{ENV})$ and the average per step loss of S_i under the j^{th} sampled environment at iteration i. The random variables Y_{ij} over all $i \in \{1 \dots T\}$ and $j \in 1, 2, \dots, m$ are all zero mean and bounded in $[-l_{max}, l_{min}]$ (here $l_{max} = 1$ as each f is bounded between $0-1$) and form a martingale in the order $Y_{11}, Y_{12}, \dots, Y_{1m}, Y_{21}, \dots, Y_{Tm}$. By Azuma-Heoffding's inequality $\frac{1}{mT}\sum_{i=1}^{T}\sum_{j=1}^{m} Y_{ij} \leq l_{max}\sqrt{\frac{2ln(\frac{1}{\delta})}{mT}} = \sqrt{\frac{2ln(\frac{1}{\delta})}{mT}}$ with probability at least $1-\delta$. Hence we obtain with probability at least $1-\delta$:

$$
\begin{aligned}
& \min_{S \in S_{1:T}} E_{d(\mathbf{ENV})}[l(\mathbf{ENV}, S)] && (30)\\
\leq\ & \frac{1}{T}\sum_{i=1}^{T} E_{d(\mathbf{ENV})}[l(\mathbf{ENV}, S)] \\
=\ & \frac{1}{T}\sum_{i=1}^{T} E_{D_i(\mathbf{ENV})}[l(\mathbf{ENV}, S)] + \frac{1}{mT}\sum_{i=1}^{T}\sum_{j=1}^{m} Y_{ij} \\
\leq\ & \frac{1}{T}\sum_{i=1}^{T} E_{D_i(\mathbf{ENV})}[l(\mathbf{ENV}, S)] + \sqrt{\frac{2ln(\frac{1}{\delta})}{mT}} \\
\leq\ & \gamma_T + \hat{l}_T^* + \sqrt{\frac{2ln(\frac{1}{\delta})}{mT}}
\end{aligned}
$$

□

□

D Proof of Proposition 2

Proposition 2. (Approximate Weak Dynamic Optimality) If *getNextEnvironment* returns examples by forward simulating beginning with a random environment and randomly choosing a new environment on reaching a goal, then consider the policy π_{mixture} that begins each new trial by choosing a list randomly from those generated throughout the T iterations of the Algorithm 1. By the no-regret property, such a mixture policy will be α approximately dynamically optimal in expectation up to an additive term $O(\sqrt{\frac{ln(1/\delta)}{T}})$ with probability greater then $1-\delta$. Further, in the (experimentally typical) case where the distribution over library sequences converges, the resulting single list is (up to approximation factor α) weakly dynamically optimal.

Proof. Consider $\pi_{mixture}$, the policy that begins each new trial of Algorithm 1 by choosing a list randomly from those generated throughout the T iterations. Let $d_{\pi_{mixture}}(\mathbf{ENV})$ be the distribution of environments encountered as a consequence of following policy $\pi_{mixture}$. Let S_i be the randomly sampled list at the i^{th} iteration. Let Y_{ij} be the difference between the expected per step loss of S_i under the distribution of $d_{\pi_i}(\mathbf{ENV})$ and the average per step loss of S_i under the j^{th} sample trajectory of horizon H. Note that here each sample j is a sequence of environments encountered as a consequence of the robot following the mixture policy $\pi_{mixture}$ whereas in Proposition 1 each j^{th} sample is an environment sampled from a distribution of environments. Following the mixture policy $\pi_{mixture}$ corresponds to randomly sampling one of the lists generated by Algorithm 1 during T iterations. The random variables Y_{ij} over all $i \in \{1 \dots T\}$ and $j \in 1, 2, \dots, m$ are all zero mean and bounded in $[-l_{max}, l_{min}]$ (here $l_{max} = 1$ as each f is bounded between $0 - 1$) and form a martingale (considering the order $Y_{11}, Y_{12}, \dots, Y_{1m}, Y_{21}, \dots, Y_{Tm}$). By Azuma-Heoffding's inequality $\frac{1}{mT} \sum_{i=1}^{T} \sum_{j=1}^{m} Y_{ij} \le l_{max} \sqrt{\frac{2ln(\frac{1}{\delta})}{mT}} = \sqrt{\frac{2ln(\frac{1}{\delta})}{mT}}$ with probability at least $1 - \delta$. Hence we obtain with probability at least $1 - \delta$:

$$\begin{aligned}
& E_{\pi_i \sim \pi_{mixture}} E_{d_{\pi_i}(\mathbf{ENV})}[l(\mathbf{ENV}, \pi_i)] \qquad (31) \\
= \; & \frac{1}{T} \sum_{i=1}^{T} E_{d_{\pi_i}(\mathbf{ENV})}[l(\mathbf{ENV}, S_i)] \\
= \; & \frac{1}{T} \sum_{i=1}^{T} E_{D_{\pi_i}(\mathbf{ENV})}[l(\mathbf{ENV}, S_i)] + \frac{1}{mT} \sum_{i=1}^{T} \sum_{j=1}^{m} Y_{ij} \\
\le \; & \frac{1}{T} \sum_{i=1}^{T} E_{D_{\pi_i}(\mathbf{ENV})}[l(\mathbf{ENV}, S_i)] + \sqrt{\frac{2ln(\frac{1}{\delta})}{mT}} \\
\le \; & \gamma_T + \hat{l}_T^* + \sqrt{\frac{2ln(\frac{1}{\delta})}{mT}}
\end{aligned}$$

□